Joins

Outer Join

Left Outer Join

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.*/

```
SELECT
    f.film_id, f.title, COUNT(*) AS num_of_copies
FROM
    film AS f
        INNER JOIN
    inventory AS i USING (film_id)
GROUP BY 1;
```

/*While you may have expected 1,000 rows to be returned (one for each film), the query returns only 958 rows. This is because the query uses an inner join, which only returns rows that satisfy the join condition. The film Alice Fantasia (film_id 14) doesn’t appear in the results, for example, because it doesn’t have any rows in the inventory table.*/

```
SELECT
    f.film_id, f.title, COUNT(*) AS num_of_copies
FROM
    film AS f
        LEFT OUTER JOIN
    inventory AS i USING (film_id)
GROUP BY 1;
```

```
SELECT
    f.film_id, f.title, COUNT(*) AS num_of_copies
FROM
    film AS f
        LEFT OUTER JOIN
    inventory AS i USING (film_id)
GROUP BY 1
HAVING num_of_copies = 1;


/*The num_copies column definition was changed from count(*) to count(i.inventory_id), which will count the number of non-null values of the inventory.inventory_id column.*/


SELECT
    f.film_id, f.title, COUNT(i.inventory_id) AS num_of_copies
FROM
    film AS f
        LEFT OUTER JOIN
    inventory AS i USING (film_id)
GROUP BY 1;


/*The join definition was changed from inner to left outer, which instructs the server to include all rows from the table on the left side of the join (film, in this case) and then include columns from the table on the right side of the join (inventory) if the join is successful.*/
```

```
SELECT
    f.film_id, f.title, i.inventory_id
FROM
    film AS f
        INNER JOIN
    inventory AS i USING (film_id)
WHERE
    f.film_id BETWEEN 13 AND 15;

SELECT
    f.film_id, f.title, i.inventory_id
FROM
    film AS f
        LEFT OUTER JOIN
    inventory AS i USING (film_id)
WHERE
    f.film_id BETWEEN 13 AND 15;
```

/*Right Outer Join

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.*/

```
SELECT
    f.film_id, f.title, i.inventory_id
FROM
    inventory as i
        RIGHT OUTER JOIN
    film AS f USING (film_id)
WHERE
    f.film_id BETWEEN 13 AND 15;
```

/*Three way outer joins - Same Logics*/

```
SELECT
    f.film_id, f.title, i.inventory_id, r.rental_date
FROM
    film f
        LEFT OUTER JOIN
    inventory i ON f.film_id = i.film_id
        LEFT OUTER JOIN
    rental r ON i.inventory_id = r.inventory_id
WHERE
    f.film_id BETWEEN 13 AND 15;
```

/*Cross Join

Cartesian product, which is essentially the result of joining multiple tables without specifying any join conditions*/

```
SELECT
    c.name category_name, l.name language_name
FROM
    category AS c
        CROSS JOIN
    language AS l;
```

/*This query generates the Cartesian product of the category and language tables, resulting in 96 rows (16 category rows × 6 language rows).*/

/*Natural Join

Natural Join allows you to name the tables to be joined but lets the database server determine what the join conditions need to be. Known as the natural join, this join type relies on identical column names across multiple tables to infer the proper join conditions.*/

```
SELECT
    c.first_name, c.last_name, DATE(r.rental_date)
FROM
    customer c
        NATURAL JOIN
    rental r;
```

```
SELECT
    cust.first_name, cust.last_name, DATE(r.rental_date)
FROM
    (SELECT
        customer_id, first_name, last_name
    FROM
        customer) cust
        NATURAL JOIN
    rental r;
```

/*Exercise - 1

Using the following table definitions and data, write a query that returns each customer

name along with their total payments:

Customer:

```
Customer_id      Name
----------- ---------------
1             John Smith
2             Kathy Jones
3             Greg Oliver
```

Payment:

```
Payment_id  Customer_id  Amount
---------- ----------- --------
101             1        8.99
102             3        4.99
103             1        7.99
```

Include all customers, even if no payment records exist for that customer.*/

```
SELECT
    c.name, SUM(p.amount)
FROM
    customer AS c
        LEFT OUTER JOIN
    payment AS p USING (customer_id)
GROUP BY 1;
```

/* Exercise - 2

Reformulate your query from Exercise 10-1 to use the other outer join type (e.g., if you used a left outer join in Exercise 10-1, use a right outer join this time) such that the results are identical to Exercise 10-1.*/

```
SELECT
    c.name, SUM(p.amount)
FROM
    payment AS p
        RIGHT OUTER JOIN
    customer AS c USING (customer_id)
GROUP BY 1;
```

/*Exercise - 3

Devise a query that will generate the set {1, 2, 3, ..., 99, 100}. (Hint: use a cross join with at least two from clause subqueries.)*/

```
SELECT ones.x + tens.x + 1
FROM
(SELECT 0 x UNION ALL
SELECT 1 x UNION ALL
SELECT 2 x UNION ALL
SELECT 3 x UNION ALL
SELECT 4 x UNION ALL
SELECT 5 x UNION ALL
SELECT 6 x UNION ALL
SELECT 7 x UNION ALL
SELECT 8 x UNION ALL
SELECT 9 x
) ones
CROSS JOIN
(SELECT 0 x UNION ALL
SELECT 10 x UNION ALL
SELECT 20 x UNION ALL
SELECT 30 x UNION ALL
SELECT 40 x UNION ALL
SELECT 50 x UNION ALL
SELECT 60 x UNION ALL
SELECT 70 x UNION ALL
SELECT 80 x UNION ALL
SELECT 90 x
) tens;
```